Searching and Sorting
Outline

1 Performance

2 Searching

3 Sorting
Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by

- its time complexity, ie, how long it takes; and
- its space complexity, ie, how much memory it needs

The running time of a program is determined by the cost of executing each statement, and the frequency of execution of each statement

Running time is expressed using an order-of-growth function of the problem size $N$, without any lower-order terms or constant coefficients
Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by

- its time complexity, ie, how long it takes; and
- its space complexity, ie, how much memory it needs

The running time of a program is determined by the cost of executing each statement, and the frequency of execution of each statement

Running time is expressed using an order-of-growth function of the problem size $N$, without any lower-order terms or constant coefficients
Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by
- its time complexity, i.e., how long it takes; and
- its space complexity, i.e., how much memory it needs

The running time of a program is determined by the cost of executing each statement, and the frequency of execution of each statement

Running time is expressed using an order-of-growth function of the problem size $N$, without any lower-order terms or constant coefficients
Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by
- its time complexity, ie, how long it takes; and
- its space complexity, ie, how much memory it needs

The running time of a program is determined by the cost of executing each statement, and the frequency of execution of each statement

Running time is expressed using an order-of-growth function of the problem size $N$, without any lower-order terms or constant coefficients
Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by

- its time complexity, ie, how long it takes; and
- its space complexity, ie, how much memory it needs

The running time of a program is determined by the cost of executing each statement, and the frequency of execution of each statement

Running time is expressed using an order-of-growth function of the problem size \( N \), without any lower-order terms or constant coefficients
Performance

Algorithms are methods for solving computational problems

Data structures are schemes for arranging data, amenable to efficient processing by algorithms

The performance characteristics of a program is determined by

- its time complexity, ie, how long it takes; and
- its space complexity, ie, how much memory it needs

The running time of a program is determined by the cost of executing each statement, and the frequency of execution of each statement

Running time is expressed using an order-of-growth function of the problem size $N$, without any lower-order terms or constant coefficients
### Performance

#### Order-of-growth classifications

<table>
<thead>
<tr>
<th>description</th>
<th>function</th>
<th>code description</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>$1$</td>
<td>statement</td>
<td>add two numbers</td>
</tr>
<tr>
<td>logarithmic</td>
<td>$\log N$</td>
<td>divide in half loop</td>
<td>binary search</td>
</tr>
<tr>
<td>linear</td>
<td>$N$</td>
<td>loop</td>
<td>find the maximum</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>divide and conquer</td>
<td>merge sort</td>
</tr>
<tr>
<td>quadratic</td>
<td>$N^2$</td>
<td>double loop</td>
<td>check all pairs</td>
</tr>
<tr>
<td>cubic</td>
<td>$N^3$</td>
<td>triple loop</td>
<td>check all triples</td>
</tr>
<tr>
<td>exponential</td>
<td>$2^N$</td>
<td>exhaustive search</td>
<td>check all subsets</td>
</tr>
</tbody>
</table>
## Performance

### Order-of-growth classifications

<table>
<thead>
<tr>
<th>description</th>
<th>function</th>
<th>code description</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>$1$</td>
<td>statement</td>
<td>add two numbers</td>
</tr>
<tr>
<td>logarithmic</td>
<td>$\log N$</td>
<td>divide in half</td>
<td>binary search</td>
</tr>
<tr>
<td>linear</td>
<td>$N$</td>
<td>loop</td>
<td>find the maximum</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>divide and conquer</td>
<td>merge sort</td>
</tr>
<tr>
<td>quadratic</td>
<td>$N^2$</td>
<td>double loop</td>
<td>check all pairs</td>
</tr>
<tr>
<td>cubic</td>
<td>$N^3$</td>
<td>triple loop</td>
<td>check all triples</td>
</tr>
<tr>
<td>exponential</td>
<td>$2^N$</td>
<td>exhaustive search</td>
<td>check all subsets</td>
</tr>
</tbody>
</table>
Performance

The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call `sys.getsizeof(x)` returns the number of bytes that a built-in object `x` consumes on a particular system.

Sizes of built-in objects on a typical system

<table>
<thead>
<tr>
<th>object</th>
<th>size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>24</td>
</tr>
<tr>
<td>float</td>
<td>24</td>
</tr>
<tr>
<td>boolean</td>
<td>24</td>
</tr>
<tr>
<td>string of <code>n</code> characters</td>
<td><code>40 + n</code></td>
</tr>
<tr>
<td>list of <code>n</code> integers</td>
<td><code>72 + 8n + 24n = 72 + 32n</code></td>
</tr>
<tr>
<td><code>m-by-n</code> list of integers</td>
<td><code>72 + 8m + m(72 + 32n) = 72 + 80m + 32mn</code></td>
</tr>
<tr>
<td>user-defined</td>
<td>hundreds of bytes, at least</td>
</tr>
</tbody>
</table>
The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call `sys.getsizeof(x)` returns the number of bytes that a built-in object `x` consumes on a particular system.

Sizes of built-in objects on a typical system

<table>
<thead>
<tr>
<th>object</th>
<th>size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>24</td>
</tr>
<tr>
<td>float</td>
<td>24</td>
</tr>
<tr>
<td>boolean</td>
<td>24</td>
</tr>
<tr>
<td>string of <code>n</code> characters</td>
<td><code>40 + n</code></td>
</tr>
<tr>
<td>list of <code>n</code> integers</td>
<td><code>72 + 8n + 24n = 72 + 32n</code></td>
</tr>
<tr>
<td><code>m</code>-by-<code>n</code> list of integers</td>
<td><code>72 + 8m + m(72 + 32n) = 72 + 80m + 32mn</code></td>
</tr>
<tr>
<td>user-defined</td>
<td>hundreds of bytes, at least</td>
</tr>
</tbody>
</table>
The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call `sys.getsizeof(x)` returns the number of bytes that a built-in object `x` consumes on a particular system.

Sizes of built-in objects on a typical system

<table>
<thead>
<tr>
<th>object</th>
<th>size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>24</td>
</tr>
<tr>
<td>float</td>
<td>24</td>
</tr>
<tr>
<td>boolean</td>
<td>24</td>
</tr>
<tr>
<td>string of $n$ characters</td>
<td>$40 + n$</td>
</tr>
<tr>
<td>list of $n$ integers</td>
<td>$72 + 8n + 24n = 72 + 32n$</td>
</tr>
<tr>
<td>$m$-by-$n$ list of integers</td>
<td>$72 + 8m + m(72 + 32n) = 72 + 80m + 32mn$</td>
</tr>
<tr>
<td>user-defined</td>
<td>hundreds of bytes, at least</td>
</tr>
</tbody>
</table>
Performance

The sizes of objects of built-in types differ from system to system, so the sizes of data types that we create also differ accordingly.

The function call `sys.getsizeof(x)` returns the number of bytes that a built-in object `x` consumes on a particular system.

Sizes of built-in objects on a typical system

<table>
<thead>
<tr>
<th>object</th>
<th>size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>24</td>
</tr>
<tr>
<td>float</td>
<td>24</td>
</tr>
<tr>
<td>boolean</td>
<td>24</td>
</tr>
<tr>
<td>string of ( n ) characters</td>
<td>( 40 + n )</td>
</tr>
<tr>
<td>list of ( n ) integers</td>
<td>( 72 + 8n + 24n = 72 + 32n )</td>
</tr>
<tr>
<td>( m )-by-( n ) list of integers</td>
<td>( 72 + 8m + m(72 + 32n) = 72 + 80m + 32mn )</td>
</tr>
<tr>
<td>user-defined</td>
<td>hundreds of bytes, at least</td>
</tr>
</tbody>
</table>
The search problem involves searching for a key in a collection of $N$ keys.

Linear search

```python
def search(key, a):
    for i, v in enumerate(a):
        if key == v:
            return i
    return -1
```

Order of growth: $N$ (linear)
The search problem involves searching for a key in a collection of $N$ keys.

Linear search

```python
def search(key, a):
    for i, v in enumerate(a):
        if key == v:
            return i
    return -1
```

Order of growth: $N$ (linear)
The search problem involves searching for a key in a collection of $N$ keys.

**Linear search**

```python
def search(key, a):
    for i, v in enumerate(a):
        if key == v:
            return i
    return -1
```

Order of growth: $N$ (linear)
The search problem involves searching for a key in a collection of $N$ keys.

**Linear search**

```python
def search(key, a):
    for i, v in enumerate(a):
        if key == v:
            return i
    return -1
```

Order of growth: $N$ (linear)
Searching

Binary search

Order of growth: \( \log N \) (logarithmic)
Searching

Binary search

successful search for the key 23
**Searching**

**Binary search**

```
<table>
<thead>
<tr>
<th>lo</th>
<th>mid</th>
<th>hi</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>29</td>
<td>33</td>
<td>48</td>
</tr>
<tr>
<td>54</td>
<td>57</td>
<td>68</td>
</tr>
<tr>
<td>77</td>
<td>84</td>
<td>98</td>
</tr>
</tbody>
</table>
```

successful search for the key 23

```
<table>
<thead>
<tr>
<th>lo</th>
<th>mid</th>
<th>hi</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>29</td>
<td>33</td>
<td>48</td>
</tr>
<tr>
<td>54</td>
<td>57</td>
<td>68</td>
</tr>
<tr>
<td>77</td>
<td>84</td>
<td>98</td>
</tr>
</tbody>
</table>
```

unsuccessful search for the key 50
Searching

Binary search

Order of growth: \( \log N \) (logarithmic)
import instream
import stdio
import sys

def _search(key, a, lo, hi):
    if hi <= lo:
        return -1
    mid = (lo + hi) // 2
    if key < a[mid]:
        return _search(key, a, lo, mid)
    elif a[mid] < key:
        return _search(key, a, mid + 1, hi)
    else:
        return mid

def search(key, a):
    return _search(key, a, 0, len(a))

def main():
    inStream = instream.InStream(sys.argv[1])
    a = inStream.readAllStrings()
    while not stdio.isEmpty():
        key = stdio.readString()
        if search(key, a) < 0:
            stdio.writeln(key)

if __name__ == '__main__':
    main()
import instream
import stdio
import sys

def _search(key, a, lo, hi):
    if hi <= lo:
        return -1
    mid = (lo + hi) // 2
    if key < a[mid]:
        return _search(key, a, lo, mid)
    elif a[mid] < key:
        return _search(key, a, mid + 1, hi)
    else:
        return mid

def search(key, a):
    return _search(key, a, 0, len(a))

def main():
    inStream = instream.InStream(sys.argv[1])
    a = inStream.readAllStrings()
    while not stdio.isEmpty():
        key = stdio.readString()
        if search(key, a) < 0:
            stdio.writeln(key)

if __name__ == '__main__':
    main()
$ more emails.txt
bob@office
carl@beach
marvin@spam
bob@office
bob@office
bob@office
mallory@spam
dave@boat
eve@airport
alice@home

$ more white.txt
alice@home
bob@office
carl@beach
dave@boat

dave@boat

$ python binarysearch.py white.txt < emails.txt
marvin@spam
mallory@spam
eve@airport
$ more emails.txt
bob@office
carl@beach
marvin@spam
bob@office
bob@office
mallory@spam
dave@boat
eve@airport
alice@home

$ more white.txt
alice@home
bob@office
carl@beach
dave@boat

dave@boat
eve@airport

$ python binarysearch.py white.txt < emails.txt
marvin@spam
mallory@spam
eve@airport
Searching

$ more emails.txt
bob@office
carl@beach
marvin@spam
bob@office
bob@office
mallory@spam
dave@boat
eve@airport
alice@home

$ more white.txt
alice@home
bob@office
carl@beach
dave@boat

$ python binarysearch.py white.txt < emails.txt
marvin@spam
mallory@spam
eve@airport
$ more emails.txt
bob@office
carl@beach
marvin@spam
bob@office
bob@office
mallory@spam
dave@boat
eve@airport
alice@home

$ more white.txt
alice@home
bob@office
carl@beach
dave@boat

$ python binarysearch.py white.txt < emails.txt
marvin@spam
mallory@spam
eve@airport
The sort problem involves rearranging a sequence of objects so as to put them in some logical order.

Insertion sort is similar to sorting a bridge hand — consider the cards one at a time, inserting each into its proper place among those already considered.

Order of growth: \( N^2 \) (quadratic)
The sort problem involves rearranging a sequence of objects so as to put them in some logical order.

Insertion sort is similar to sorting a bridge hand — consider the cards one at a time, inserting each into its proper place among those already considered.

Order of growth: \( N^2 \) (quadratic)
The sort problem involves rearranging a sequence of objects so as to put them in some logical order.

Insertion sort is similar to sorting a bridge hand — consider the cards one at a time, inserting each into its proper place among those already considered.

Order of growth: $N^2$ (quadratic)
The sort problem involves rearranging a sequence of objects so as to put them in some logical order.

Insertion sort is similar to sorting a bridge hand — consider the cards one at a time, inserting each into its proper place among those already considered.

Order of growth: $N^2$ (quadratic)
import stdio
import sys

def exchange(a, i, j):
    a[i], a[j] = a[j], a[i]

def sort(a):
    n = len(a)
    for i in range(1, n):
        j = i
        while (j > 0) and (a[j] < a[j - 1]):
            exchange(a, j - 1, j)
            j -= 1

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    main()

$ more tiny.txt
the and was his you tom but for had him

$ python insertion.py < tiny.txt
and but for had him his the tom was you
import stdio
import sys

def exchange(a, i, j):
    a[i], a[j] = a[j], a[i]

def sort(a):
    n = len(a)
    for i in range(1, n):
        j = i
        while (j > 0) and (a[j] < a[j - 1]):
            exchange(a, j - 1, j)
            j -= 1

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    main()
import stdio
import sys

def exchange(a, i, j):
a[i], a[j] = a[j], a[i]

def sort(a):
n = len(a)
for i in range(1, n):
j = i
while (j > 0) and (a[j] < a[j - 1]):
    exchange(a, j - 1, j)
j -= 1

def main():
a = stdio.readAllStrings()
sort(a)
for s in a:
    stdio.write(s + ', ')
stdio.writeln()

if __name__ == '__main__':
    main()

$ more tiny.txt
the and was his you tom but for had him

$ python insertion.py < tiny.txt
and but for had him his the tom was you
import stdio
import sys

def exchange(a, i, j):
    a[i], a[j] = a[j], a[i]

def sort(a):
    n = len(a)
    for i in range(1, n):
        j = i
        while (j > 0) and (a[j] < a[j - 1]):
            exchange(a, j - 1, j)
            j -= 1

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    main()

$ more tiny.txt
the and was his you tom but for had him

$ python insertion.py < tiny.txt
and but for had him his the tom was you
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a[0]</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j</td>
<td>O</td>
<td>S</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Sorting

### Trace

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>O</td>
<td>S</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>A</td>
<td>E</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>A</td>
<td>E</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>A</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>E</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
</tbody>
</table>

```python
a[]
```
Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list.

To sort a list, we divide it into two halves, sort the two halves recursively, and then merge the results.

Order of growth: $N \log N$ (linearithmic)
Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list.

To sort a list, we divide it into two halves, sort the two halves recursively, and then merge the results.

Order of growth: $N \log N$ (linearithmic)
Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list.

To sort a list, we divide it into two halves, sort the two halves recursively, and then merge the results.

```
input  M E R G E S O R T  T E X A M P L E
sort left half  E E G M O R R S  T E X A M P L E
sort right half  E E G M O R R S  A E E L M P T X
merge results  A E E E E G L M  M O P R R S T X
```

Order of growth: \( N \log N \) (linearithmic)
Merge sort is based on a simple operation known as merging: combining two ordered lists to make one larger ordered list.

To sort a list, we divide it into two halves, sort the two halves recursively, and then merge the results.

<table>
<thead>
<tr>
<th>input</th>
<th>MERGESORT</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort left half</td>
<td>E E G M O R R S</td>
<td>TEXAMPLE</td>
</tr>
<tr>
<td>sort right half</td>
<td>E E G M O R R S</td>
<td>A E E L M P T X</td>
</tr>
<tr>
<td>merge results</td>
<td>A E E E E G L M</td>
<td>M O P R R S T X</td>
</tr>
</tbody>
</table>

Order of growth: \( N \log N \) (linearithmic)
merge.py: Read strings from standard input, sort them into increasing order, and write them to standard output.

```python
import stdarray
import stdio
import sys

def _merge(a, lo, mid, hi, aux):
    n = hi - lo
    i = lo
    j = mid
    for k in range(n):
        if i == mid:
            aux[k] = a[j]
            j += 1
        elif j == hi:
            aux[k] = a[i]
            i += 1
        elif a[j] < a[i]:
            aux[k] = a[j]
            j += 1
        else:
            aux[k] = a[i]
            i += 1
    a[lo:hi] = aux[0:n]
```
Sorting

merge.py: Read strings from standard input, sort them into increasing order, and write them to standard output.

```python
import stdarray
import stdio
import sys

def _merge(a, lo, mid, hi, aux):
    n = hi - lo
    i = lo
    j = mid
    for k in range(n):
        if i == mid:
            aux[k] = a[j]
            j += 1
        elif j == hi:
            aux[k] = a[i]
            i += 1
        elif a[j] < a[i]:
            aux[k] = a[j]
            j += 1
        else:
            aux[k] = a[i]
            i += 1
    a[lo:hi] = aux[0:n]
```
```
def _sort(a, lo, hi, aux):
    n = hi - lo
    if n <= 1:
        return
    mid = (lo + hi) // 2
    _sort(a, lo, mid, aux)
    _sort(a, mid, hi, aux)
    _merge(a, lo, mid, hi, aux)

def sort(a):
    n = len(a)
    aux = stdarray.create1D(n)
    _sort(a, 0, n, aux)

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ', ')
    stdio.writeln()

if __name__ == '__main__':
    main()
```

```
$ more tiny.txt
the and was his you tom but for had him

$ python merge.py < tiny.txt
and but for had him his the tom was you
```
def _sort(a, lo, hi, aux):
    n = hi - lo
    if n <= 1:
        return
    mid = (lo + hi) // 2
    _sort(a, lo, mid, aux)
    _sort(a, mid, hi, aux)
    _merge(a, lo, mid, hi, aux)

def sort(a):
    n = len(a)
    aux = stdarray.create1D(n)
    _sort(a, 0, n, aux)

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    main()
Sorting

def _sort(a, lo, hi, aux):
    n = hi - lo
    if n <= 1:
        return
    mid = (lo + hi) // 2
    _sort(a, lo, mid, aux)
    _sort(a, mid, hi, aux)
    _merge(a, lo, mid, hi, aux)

def sort(a):
    n = len(a)
    aux = stdarray.create1D(n)
    _sort(a, 0, n, aux)

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ', ')
    stdio.writeln()

if __name__ == '__main__':
    main()

$ more tiny.txt
the and was his you tom but for had him

$ python merge.py < tiny.txt
and but for had him his the tom was you
Sorting

def _sort(a, lo, hi, aux):
    n = hi - lo
    if n <= 1:
        return
    mid = (lo + hi) // 2
    _sort(a, lo, mid, aux)
    _sort(a, mid, hi, aux)
    _merge(a, lo, mid, hi, aux)

def sort(a):
    n = len(a)
    aux = stdarray.create1D(n)
    _sort(a, 0, n, aux)

def main():
    a = stdio.readAllStrings()
    sort(a)
    for s in a:
        stdio.write(s + ', ')
    stdio.writeln()

if __name__ == '__main__':
    main()

$ more tiny.txt
the and was his you tom but for had him

$ python merge.py < tiny.txt
and but for had him his the tom was you
Sorting

Trace (_merge())
### Sorting

**Trace (_merge())**

<table>
<thead>
<tr>
<th>k</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>copy</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>R</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**merged result**

A C E E E G M R R T

E E G M R A C E R T

A C E E E G M R R T
Sorting

Trace (_sort())

_merge(a, 0, 0, 1, aux)
_merge(a, 2, 2, 3, aux)
_merge(a, 0, 1, 3, aux)
_merge(a, 4, 4, 5, aux)
_merge(a, 6, 6, 7, aux)
_merge(a, 4, 5, 7, aux)
_merge(a, 0, 3, 7, aux)
_merge(a, 8, 8, 9, aux)
_merge(a, 10, 10, 11, aux)
_merge(a, 8, 9, 11, aux)
_merge(a, 12, 12, 13, aux)
_merge(a, 14, 14, 15, aux)
_merge(a, 12, 13, 15, aux)
_merge(a, 8, 11, 15, aux)
_merge(a, 0, 7, 15, aux)
Sorting

Trace (_sort())

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>M</td>
<td>E</td>
<td>R</td>
<td>G</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>M</td>
<td>G</td>
<td>R</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>E</td>
<td>L</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>P</td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
</tbody>
</table>

_merge(a, 0, 0, 1, aux)
_merge(a, 2, 2, 3, aux)
_merge(a, 0, 1, 3, aux)
_merge(a, 4, 4, 5, aux)
_merge(a, 6, 6, 7, aux)
_merge(a, 4, 5, 7, aux)
_merge(a, 0, 3, 7, aux)
_merge(a, 8, 8, 9, aux)
_merge(a, 10, 10, 11, aux)
_merge(a, 8, 9, 11, aux)
_merge(a, 12, 12, 13, aux)
_merge(a, 14, 14, 15, aux)
_merge(a, 12, 13, 15, aux)
_merge(a, 8, 11, 15, aux)
_merge(a, 0, 7, 15, aux)
import stdio
import sys
from counter import Counter

def main():
    words = stdio.readAllStrings()
    words.sort()
    zipf = []
    for i in range(len(words)):
        if (i == 0) or (words[i] != words[i - 1]):
            entry = Counter(words[i], len(words))
            zipf += [entry]
            zipf[len(zipf) - 1].increment()
    zipf.sort()
    zipf.reverse()
    for entry in zipf:
        stdio.writeln(entry)

if __name__ == '__main__':
    main()

$ python frequencycount.py < tomsawyer.txt
the: 3452
and: 2908
a: 1758
to: 1741
of: 1539
was: 1124
in: 926
...
import stdio
import sys
from counter import Counter

def main():
    words = stdio.readAllStrings()
    words.sort()
    zipf = []
    for i in range(len(words)):
        if (i == 0) or (words[i] != words[i - 1]):
            entry = Counter(words[i], len(words))
            zipf += [entry]
            zipf[len(zipf) - 1].increment()
    zipf.sort()
    zipf.reverse()
    for entry in zipf:
        stdio.writeln(entry)

if __name__ == '__main__':
    main()

$ python frequencycount.py < tomsawyer.txt
the: 3452
and: 2908
a: 1758
to: 1741
of: 1539
was: 1124
in: 926
...
import stdio
import sys
from counter import Counter

def main():
    words = stdio.readAllStrings()
    words.sort()
    zipf = []
    for i in range(len(words)):
        if (i == 0) or (words[i] != words[i - 1]):
            entry = Counter(words[i], len(words))
            zipf += [entry]
            zipf[len(zipf) - 1].increment()
    zipf.sort()
    zipf.reverse()
    for entry in zipf:
        stdio.writeln(entry)

if __name__ == '__main__':
    main()

$ python frequencycount.py < tomsawyer.txt
the: 3452
and: 2908
a: 1758
to: 1741
of: 1539
was: 1124
in: 926
...